Totally and Partially Ordered Hierarchical Planners in PDDL4J Library

Damien Pellier, Humbert Fiorino
Univ. Grenoble Alpes - LIG
F-38000 Grenoble, France
{Damien.Pellier, Humbert.Fiorino}@imag.fr

Abstract

In this paper, we outline the implementation of the TFD (Totally Ordered Fast Downward) and the PFD (Partially ordered Fast Downward) hierarchical planners that participated in the first HTN IPC competition in 2020. These two planners are based on forward-chaining task decomposition coupled with a compact grounding of actions, methods, tasks and HTN problems.

Introduction

The TFD (Totally Ordered Fast Downward) and PFD (Partial ordered Fast Downward) hierarchical planners are based on forward-chaining task decomposition used by the SHOP2 planner (Nau et al. 2003) coupled with a compact grounding of actions, methods, tasks and HTN problems. Both planners accept as input HDDL (Hierarchical Domain Description Language) proposed by (Höller et al. 2020) and are implemented on top of the PDDL4J library (Pellier and Fiorino 2018). In this short paper we present first the compact representation used by TFD and PFD as well as the grounding procedure implemented. Finally, we conclude with a brief presentation of the search strategy implemented in both planners.

Grounding technique

Most modern planners work with grounded representations of the planning problem. However, planning domains are commonly defined with a lifted description language such as PDDL (Ghallab et al. 1998) or HDDL (Höller et al. 2020). Thus, planning systems have to generate a grounded representation of the lifted domain in a preprocessing step, the objective of which is to generate the most compact grounded representation possible without removing any action, method or fluent needed for a solution plan. The more compact the grounded representation is, the more efficient is the search for a solution plan as reducing the size of the search space speeds up search and heuristic value computation. In practice, computing a grounded representation from a lifted representation is quite straightforward. All possible instantiations of ground predicates, primitive actions, abstract tasks and methods must be computed, and appropriately replaced by their ground versions in the lifted representation.

In the context of classical (non hierarchical) planning, the planners FF (Hoffmann and Nebel 2001) and FastDownward (Helmert 2006) have implemented techniques for transforming lifted to ground planning representation that are still used in many planners today. Regarding hierarchical planning, (Ramoul et al. 2017) inspired from (Koehler and Hoffmann 1999) have been the first to propose an efficient grounding preprocessing, and it was successfully applied to the planners proposed by (Schreiber et al. 2019). Recently, (Behnke et al. 2020) has proposed novel techniques.

In the TFD and PFD planners, the grounding combines the approches proposed by (Ramoul et al. 2017) and (Behnke et al. 2020). It is based on 6 steps:

1. Encode the lifted domain into an integer representation,

2. Simplify the lifted representation and infer types from predicates,

3. Instantiate the set of actions by removing unreachable actions with respect to the inertia principle (Koehler and Hoffmann 1999),

4. Filter out actions grounded in reachability analysis (Hoffmann and Nebel 2001),

5. Instantiate the set of methods by removing unuseful methods with respect to the inertia principle, and by recursively decomposing the initial tasks network of the planning problem. Tasks that are not reachable, i.e., tasks for which there are no relevant ground actions or methods are pruned.

6. Encode the actions and the methods into bitset representation.

More details on the implementation of the different instantiation techniques are available in PDDL4J opensource repository: `https://github.com/pellierd/pddl4j`.

Search procedures

The non-deterministic TFD procedure for solving a HTN planning problem is given in Algorithm 1. This procedure is based directly on the recursive definition of a solution plan for HTN planning problems.

The TFD procedure takes as input a problem $P = (s_0, T, A, M)$ where s_0 is the initial state, $T =$

$\langle t_1, t_2, ..., t_k \rangle$ is a list of tasks, A the set of actions, and M the set of methods, all in their ground representation. First, the procedure tests if the list of tasks T is empty (line 2). In this case, no task has to be executed, thus the empty plan is returned. Then the procedure gets the first task t_1 of the list T. Two cases must be considered depending on the type of t_1:

Case 1. If t_1 is primitive (line 3) then the procedure computes the set of all the ground actions that accomplishes t_1 and that are applicable in s_0 (line 4). If there is no action (line 5), the procedure fails because no action accomplishes the goal task t_1. Then the procedure non-deterministically chooses an action that accomplishes the task (line 6), and calls itself recursively on the planning problem $P' = (\gamma(s_0, a_1), T - \{t_1\}, A, M)$ (line 7). Finally, if the recursive call to the procedure fails to find a plan π, it returns failure (line 8); otherwise it returns the plan that is the concatenation of a and π (line 9).

Case 2. If t_1 is non-primitive (line 10) then the procedure computes the set of ground decompositions that accomplish t_1 and that are applicable in s_0 (line 11). If there is no decomposition to accomplish t_1 (line 12) then the procedure returns failure. Otherwise the procedure non-deterministically chooses a decomposition d that accomplishes the task t_1 (line 13), and recursively returns the solution plan for the problem $P' = (s_0, subtasks(d) \oplus \langle t_2, \ldots, t_k \rangle, A, M)$ (line 14).

Practically non-deterministic choices are made by systematically choosing the task networks with the least amount of non-decomposed tasks. In the case where several networks have the same number of tasks remaining to be decomposed, the task network containing the least number of actions is chosen.

The search procedure implemented in PFD is almost similar. The main difference is no longer to choose the first t_1 task in the task network but to choose the first task that does not have any predecessor task in the task network. In addition, each time case 2 applies, it is necessary to check the consistency of the ordering constraints of the task network in order to generate a-cyclic task networks. This check is performed before line 14 by calculating the transitive closure of the ordering constraints. The computation of the transitive closure is based on Warshall algorithm. The complexity is $O(n^3)$ where n is the number of tasks of the task network.

References

[Behnke et al. 2020] Behnke, G.; Höller, D.; Schmid, A.; Bercher, P.; and Biundo, S. 2020. On succinct groundings of HTN planning problems. In *The AAAI Conference on Artificial Intelligence*, 9775–9784. AAAI Press.

[Ghallab et al. 1998] Ghallab, M.; Howe, A.; Knoblock, G.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. *PDDL: The Planning Domain Definition Language*. Artificial Intelligence Planning Systems.

[Helmert 2006] Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Algorithm 1: TFD(s_0, T, A, M)

1 Let $T = \langle t_1, \ldots, t_k \rangle$
2 **if** $k = 0$ **then** return the empty plan $\langle \rangle$
3 **if** t_1 *is primitive* **then**
4 $\quad$ $A' \leftarrow$ the set of relevant actions for t_1 and applicable in s_0
5 $\quad$ **if** $A' = \emptyset$ **then return** failure
6 $\quad$ non-deterministically choose an action $a \in A'$
7 $\quad$ $\pi \leftarrow TFD(\gamma(s_0, a), \langle t_2, \ldots, t_k \rangle, A, M)$
8 $\quad$ **if** $\pi =$ **failure then** return **failure**
9 $\quad$ **else return** $a \oplus \pi$
10 **else if** t_1 *is a non-primitive task* **then**
11 $\quad$ $M' \leftarrow$ the set of relvant methods for t_1 and applicable in s_0
12 $\quad$ **if** $M' = \emptyset$ **then return** failure
13 $\quad$ non-deterministically choose a decomposition $m \in M'$
14 $\quad$ **return** TFD($s_0, subtasks(d) \oplus \langle t_2, \ldots, t_k \rangle, A, M$)

[Hoffmann and Nebel 2001] Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 253–302.

[Höller et al. 2020] Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2020. HDDL: an extension to PDDL for expressing hierarchical planning problems. In *The AAAI Conference on Artificial Intelligence*, 9883–9891. AAAI Press.

[Koehler and Hoffmann 1999] Koehler, J., and Hoffmann, J. 1999. Handling of inertia in a planning system. Technical report.

[Nau et al. 2003] Nau, D. S.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: an HTN planning system. *J. Artif. Intell. Res.* 20:379–404.

[Pellier and Fiorino 2018] Pellier, D., and Fiorino, H. 2018. PDDL4J: a planning domain description library for Java. *J. Exp. Theor. Artif. Intell.* 30(1):143–176.

[Ramoul et al. 2017] Ramoul, A.; Pellier, D.; Fiorino, H.; and Pesty, S. 2017. Grounding of HTN planning domain. *Int. J. Artif. Intell. Tools* 26(5):1760021:1–1760021:24.

[Schreiber et al. 2019] Schreiber, D.; Pellier, D.; Fiorino, H.; and Balyo, T. 2019. Tree-rex: SAT-based tree exploration for efficient and high-quality HTN planning. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, 382–390.